Ad Hoc Grid Security Infrastructure

Kaizar Amin*[†], Gregor von Laszewski*[‡][§], Mikhail Sosonkin[¶],
Armin R. Mikler[†], Mihael Hategan[‡]
*Mathematics and Computer Science Division, Argonne National Laboratory, U.S.A.
[†]Computer Science and Engineering Department, University of North Texas, U.S.A.
[‡]Computation Institute, University of Chicago, U.S.A.
[¶]Computer Science Department, Polytechnic University, U.S.A.
[§]Corresponding author: gregor@mcs.anl.gov

http://www.mcs.anl.gov/˜gregor/papers/vonLaszewski-adhoc-security.pdf

***Abstract*— This paper describes an ad hoc Grid security infrastructure developed as a part of the Java CoG Kit project. It supports several requirements specific to the sporadic nature of ad hoc Grids. It focuses on identity management, identity verification, and authorization control in spontaneous Grid collaborations without pre-established policies or environments. It adopts established community standards, with modifications where needed. This paper also discusses the integration of the ad hoc Grid security infrastructure in an ad hoc Grid implementation. The implementation supports secure collaboration in ad hoc Grids using commodity technologies such as the Java CoG Kit, JXTA, GSI, and XACML.**

I. Introduction

The need for *sporadic* or *ad hoc Grids* has its origin in the development of an infrastructure supporting scientific experiment management for Grand Challenge Applications [1], [2]. As part of this development, von Laszewski identified the important differences between compute center maintained Grids [3] and Grids that are maintained based on sporadic interactions and use patterns [4] in an ad hoc fashion. This includes spontaneous, and time limited exposure of the infrastructure and the integration, deployment [5], administration [6], and usage of resources (including the users) and services as part of an experiment management infrastructure. We termed such an infrastructure *sporadic Grid* [4] or *ad hoc Grid*.

An ad hoc Grid allows Grid entities, also referred to as ad hoc Grid peers, to spontaneously establish an ad hoc relationship, join existing Grids, dynamically contribute services to the Grid, and invoke services offered by other peers in the Grid. Ad hoc Grids facilitate interaction in an autonomous fashion without requiring pre-configured environments or management policies. They support a large class of applications that cannot be conventionally supported by traditional Grid environments. These applications include market-oriented applications, transient collaborations, sporadic interactions, and other community applications that require on-the-fly Grid establishment and deployment [7]. Ongoing research within the Java CoG Kit project [8] is focusing on realizing such dynamic, autonomous, self-adaptive, self-managing, and community-controlled Grid infrastructure.

Tightly controlled and systematically enforced security frameworks [9] have encouraged the adoption of Grids by the scientific and commercial communities. In traditional Grid environments, every entity has a pre-established trust relationship with a central administrative authority. For every entity, this authority assigns a unique Grid identity and a set of authorization privileges within the scope of the established trust. Within the realm of these assigned identities and privileges, Grid entities can seamlessly collaborate and interact with each other. These interactions are impartially monitored by the administrators. In the event of policy violations, the administrators terminate their trust relationship with the violating entity and revoke its Grid usage privileges.

Such security patterns cannot be applied in the context of ad hoc Grids. Ad hoc Grids facilitate structural independence, whereby they do not critically rely on the existence of any particular entity. The unavailability of a Grid peer may result in the unavailability of services hosted on that peer. However, it does not result in a non functional Grid. For example, in traditional Grids, the unavailability of the registration service results in a non functional Grid because other Grid entities cannot discover the existing services. Although redundancy and replication of critical services help in improving its fault tolerance, they merely provide a partial solution, without completely eliminating the dependence on external entities. In ad hoc Grids, no critical service is hosted on a single resource or a group of resources. Instead, *all* members of the ad hoc Grid equally participate in the realization of critical Grid services, enabling them to be independent of the availability of a specific peer or group of peers. Ad hoc Grids also support control independence. Control independence in an ad hoc Grids reflects its ability to manage its security and usage policies in the absence of a central controller. Because of its structural independence, peers in an ad hoc Grid cannot rely on external support for crucial security enforcement services. Thus, the centralized administrative services in traditional Grids that are responsible for membership, access, and usage control on Grid resources are segregated to be hosted on every participating peer. Every entity in an ad hoc Grid is responsible for maintaining and securing itself. Depending on their individual policies, participants may allow universal access or restrict access to a few trusted peers. Nevertheless, without an integrated ad hoc Grid security infrastructure that offers the appropriate tools and security semantics, these independent security policies can lead to either major security compromises or complete non interaction between ad hoc Grid peers.

Motivated by the need to support structure independence and control independence in ad hoc Grids without any compromise in security, we have designed an ad hoc Grid security infrastructure (AGSI) within the Java CoG Kit project. The rest of this paper is organized as follows. Section II gives an overview of the authentication subsystem of AGSI that enables peers to establish and verify Grid identities. Section III discusses the authorization subsystem of AGSI that allows peers to autonomously formulate and enforce individual security policies. Section IV discusses the integration of AGSI in a prototype ad hoc Grid implementation. Section V describes several community-based security frameworks discussed in literature. Section VI summarizes this paper.

II. The Authentication Subsystem

The authentication subsystem in AGSI offers the semantics necessary to establish, maintain, and verify peer identities in an ad hoc Grid. The authentication subsystem of AGSI is reusing concepts found in the Grid Security Infrastructure (GSI) [10]. However, it includes several modifications.

Every peer in an ad hoc Grid is uniquely identified by an identity. Following the GSI model, a peer identity is represented as an X.509 public key certificate [11]. Therefore, every peer identity is approved (digitally signed) by a certificate authority (CA). Identities can be signed by commercially available CAs [12] or by community-established CAs [13]. Additionally, a peer may choose to act as a CA generating identities for other peers. A peer is at the liberty to establish multiple identities generated by different CAs. It is free to choose any CA to establish its identity. AGSI does not enforce the usage of any particular CA, nor does it give one CA preference over another.

Independent of the CAs chosen to establish its own identities, every peer has a set of trusted CAs. A CA trusted by a peer implies that the peer will recognize and honor all identities generated by that CA. Let $I_x = \bigcup_{k=1}^{n} i_x^{c_k}$ be a set of identities established by a peer x, where $i_x^{c_k}$ denotes the peer identity i_x issued by the CA c_k. Let $C_x = \bigcup_{j=1}^{m} c_j$ be the set of CAs trusted by peer x. A mutually authenticated transaction is permitted between ad hoc peers x and $y \iff \exists i_x^{c_a} \in I_x$ and $\exists i_y^{c_b} \in I_y$ such that $c_a \in C_y$ and $c_b \in C_x$.

Peer identities can be maintained within the local file system. Private keys associated with the X.509 public key certificates are protected on the local file system by encrypting them with a pass-phrase. However, it would be impractical for peers to explicitly provide the pass-phrase decrypting the private key for every Grid transaction. Therefore, to support single-sign-on solutions, we use GSI-based X.509 proxy certificates. Like public key certificates, proxy certificates bind a unique public key to a subject name. Unlike public key certificates, however, the issuer of proxy certificate is identified by a public key certificate or another proxy certificate rather than a CA certificate. Hence, proxy certificates can be created on the fly without requiring any intervention from conventional CAs. Using its private key, the peer generates a proxy certificate with a limited lifetime. The newly generated proxy certificate and its private key are maintained within the local file system. For all subsequent Grid interactions, the peer authenticates itself using the proxy certificate rather than its public key certificate. Since the private key associated with the proxy certificate is protected by the local file system permissions rather than encrypting it, no manual response is required by the peer; hence, single-sign-on. Further, since the proxy certificate has a short lifetime, it is typically permissible to protect it in a less secure manner than the long-term private key. Although GSI proposes and actively uses proxy certificates for dynamic credential delegation [14], AGSI does not permit credential delegation. In the absence of a pre-established trust relationship, it is impractical and insecure for any peer to delegate a subset of its credentials to any other peer in the ad hoc Grid. Therefore, AGSI uses X.509 proxy certificates for the sole purpose of repeated authentication without any support for dynamic delegation.

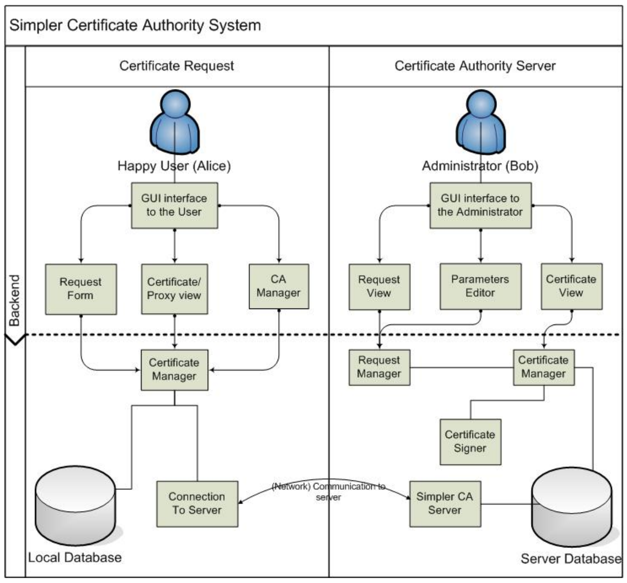


Fig. 1. High-level SimplerCA design

As discussed earlier, every peer in an ad hoc Grid can become a CA generating identities for other peers. There can be several motivations for an ad hoc Grid peer to generate identities for other peers. The certificate-signing peer can be a commercial CA offering its certificate-signing services within the Grid. Likewise, a service-providing peer may trust only peer identities generated by itself. CA management is a nontrivial task without user-friendly CA management tools. To assist ad hoc Grid peers with the complex task of issuing and maintaining Grid credentials, AGSI offers a personal certificate management system, referred to as the *SimplerCA* system [6].

Figure 1 shows the high-level design of the system

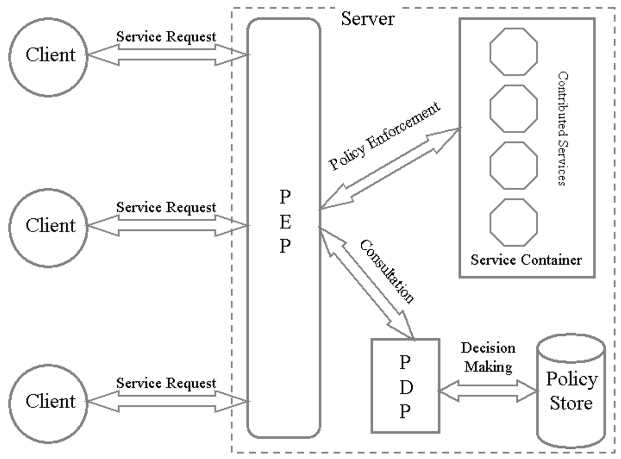


Fig. 2. Policy-based service authorization is AGSI

and depicts interactions between the certificate requesting peer and the certificate issuing peer. A requesting peer requests a certificate from an issuer using the SimplerCA client interface. The client interface contacts the certificate server and retrieves all the attributes required for the certificate. These attributes are defined by the issuer based on its internal CA policy. Once the requester completes the required information and approves its submission, the client interface forwards the information to the client certificate manager. The client certificate manager saves the information in a secure local database and generates the certificate request. The client certificate manager then uses a connection to the CA server to send the certificate request to the issuer. The CA server accepts the request and saves it in the server database. The certificate-issuing peer can evaluate the certificate request using the SimplerCA server interface. At this point the issuer can either sign the request and generate a certificate for the peer requesting it or deny it. If the certificate issuer signs the request, the certificate is saved in the server database. The certificate issuer might notify the requester that the certificate is ready through some other agreed-on protocol (such as email), or the requester may periodically poll the issuer for a signed certificate.

III. The Authorization Subsystem

Although the authentication subsystem provides a framework to establish, maintain, and verify peer identities, it does not associate trust relationships with these identities. In other words, an authenticated peer does not necessarily imply a trustworthy peer. Thus, the authorization subsystem of AGSI complements its authentication system by offering an autonomous framework for peers to control their Grid environments. Rather than a single Grid authorization policy, AGSI supports a distributed and fragmented Grid policy whereby each policy fragment is systematically controlled and enforced by different peers participating in the ad hoc Grid. Several popular traditional Grid frameworks [9] offer tightly bound static mapping between Grid users and their privileges. In such systems, Grid administrators evaluate every individual Grid user and accordingly map it to a specific set of privileges. Explicit evaluation and configuration of authorization privileges for every participating entity results in an impractical and non-scalable solution in ad hoc Grids with sporadic collaborations and continuously changing members. Therefore, the authorization subsystem offered in AGSI supports a scalable and maintainable policy formulation in ad hoc Grids that is agnostic to the constant flux in Grid environments.

Listing 1. Sample XACML policy fragment showcasing the rule elements of a policy

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy
xmlns="urn:oasis:names:tc:xacml:1.0:policy"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
PolicyId="AdHocGridPolicy"
RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:ordered-permit-overrides">

  <Description>
    This policy applies to all users of the ad hoc Grid trying to
    access the service named http://myExecuitonService.cogkit.org.
    It allows access to all methods of the protected service by the
    members of the Java CoG Kit group. All other users are denied any
    access.
  </Description>

  <Target>
    <Subjects><AnySubject/></Subjects>
    <Resources>
      <ResourceMatch
        MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
          http://myExecutionService.cogkit.org/
        </AttributeValue>
        <ResourceAttributeDesignator
          DataType="http://www.w3.org/2001/XMLSchema#anyURI"
          AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
      </ResourceMatch>
    </Resources>
    <Actions><AnyAction/></Actions>
  </Target>

  <Rule RuleId="CoGKitRule" Effect="Permit">
    <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <Apply
        FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
        <SubjectAttributeDesignator
          DataType="http://www.w3.org/2001/XMLSchema#string"
          AttributeId="group"
          Issuer="admin@adhoc.cogkit.org"/>
      </Apply>
      <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">
        Java CoG Kit Group
      </AttributeValue>
    </Condition>
  </Rule>

  <!-- Other rules can come here -->

  <Rule RuleId="FinalRule" Effect="Deny"/>
</Policy>
```

The most critical element of an authorization system is its policy language. AGSI adopts the standards-based XACML (eXtensible Access Control Markup Language) authorization and access control policy language [15]. XACML policies allow users to conveniently express access control criteria and requirements with a general-purpose access control policy language written in XML. XACML offers AGSI a simplified, yet comprehensive, environment to autonomously express and enforce distributed authorization policies independent of any external support.

Figure 2 describes a typical setup enforcing authorization policies in the pull sequence [16]. Every peer contains a policy enforcement point (PEP) protecting a set of contributed services. All invocations and access requests to the services are intercepted by the PEP. Based on the service usage request and accompanying peer credentials, the PEP, formulates a *request context* and passes it to the policy decision point (PDP). The PDP inspects the request and identifies a relevant policy corresponding to the target service. Based on the data provided in the request and the rules specified in the policy, the PDP makes a decision whether to *permit* or *deny* the requested access. The evaluated decision is conveyed to the PEP which then enforces the access control. Although the XACML specification permits the PEP and PDP to be hosted on different machines, AGSI hosts both components on the same ad hoc Grid peer.

XACML is a comprehensive and feature-rich language. Although a complete description of XACML is beyond the scope of this paper, we describe some of the basic elements used in AGSI. An XACML policy, as used in AGSI, contains a root *policy* element. Every XACML policy has an associated *target* element and a set of *rule* elements. The target elements assists the PDP in selecting the most appropriate policy for the given request context. The target element encapsulates the *subject*, *resource*, and *action* elements. The subject refers to the identity of the service requester. It allows the PDP to select a policy based on various attributes associated with the requesting subject. The resource represents the target ad hoc Grid service. It allows the PDP to conveniently map an authorization policy with a particular service. The action of a policy target specifies the methods of a contributed service that need to be enforced by the given policy. If all the conditions of the target element are satisfied, the PDP utilizes that policy for its decision.

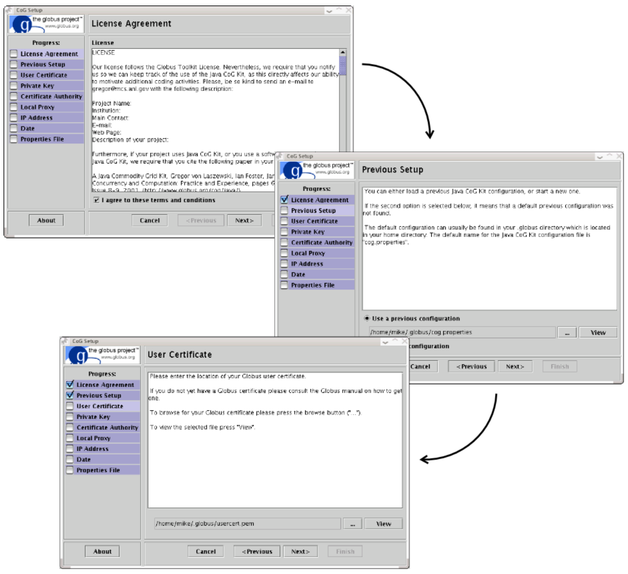
Once the PDP maps a request context to a particular policy, it evaluates all the rules associated with that policy. The policy can have any number of rules that contain the core logic of the XACML policy. The rule element represents a Boolean condition. It also has an *effect* attribute (permit, deny, indeterminate, or not applicable) that is returned when the condition associated with rule evaluates to true. Further, XACML specification defines a suite of rule combining algorithms that allow the PDP to combine the effects of multiple rules within a policy into a single effect corresponding to that policy. Several rules such as deny overrides, permit overrides, first applicable, and only one applicable are supported. Attribute values within the policy logic are resolved by using the *AttributeDesignator* and *AttributeSelector* elements. The AttributeDesignator references values by the attribute identifier, data type, and other metadata, whereas the AttributeSelector element uses XPath queries to resolve attribute values. Listing 1 shows a sample XACML policy fragment specifying that the given policy be enforced for *all* users trying to access *any* method of the *MyExecutionService*. It permits all the members of the Java CoG Kit Grid to access the MyExecutionService. All other service requests are denied. For a detailed understanding of all the elements supported by XACML and their corresponding semantics, the reader is directed to [17].

IV. Implementation

To validate our model for AGSI and to verify its utility and flexibility for ad hoc Grids, we have integrated AGSI in a prototype ad hoc Grid implementation. The implementation allows participating peers to establish spontaneous collaborations. Peers can create new ad hoc Grid communities, discover existing communities, join discovered communities, and communicate with peers in a community. Our implementation operates in a distributed environment independent of any particular Grid entity (structure independence). We have implemented the community management framework using the JXTA technology [18]. JXTA is a collection of open peer-to-peer protocols and services that allow any device with a "network heartbeat" to communicate and collaborate with other Jxta peers autonomously. It provides a mechanism to create virtual ad hoc collaborations without exposing any of the underlying peer-to-peer protocol complexities. It enables the formation of a self-organizing super-peer-based overlay network on the Internet. Figure 3 shows the user interface for the ad hoc Grid creation, discovery, presence management, and group communication.

The ad hoc Grid architecture assumes a service-oriented environment, where resources are contributed and consumed as services. We do not impose any restrictions on the nature of contributed services. These can represent a single compute or data sources, or they can represent the entire computational cluster or data storage system. Additionally, these contributed services could be potentially shared with existing static Grids or other community applications.

Peers participating in an ad hoc Grid can contribute services to be invoked by other members of the community. These services are advertised by providers in the form of service advertisements containing important details such as the peer identity, service name, service description, service contact, and qualitative attributes of the service expressed as ClassAd [19] constructs. Figure 4 shows the user interface to publish and discover service advertisements.



LICENSE

Our license follows the Globus Toolkit License. Nevertheless, we require that you notify us so we can keep track of the use of the Java CoG Kit, as this directly affects our ability to motivate additional coding activities. Please, be so kind to send an e-mail to gregor@mcs.anl.gov with the following description:

Project Name:
Institution:
Main Contact:
E-mail:
Web Page:
Description of your project:

Furthermore, if your project uses Java CoG Kit, or you use a softw
Java CoG Kit, we require that you cite the following paper in your

A Java Commodity Grid Kit, Gregor von Laszewski, Ian Foster, Jar
Concurrency and Computation: Practice and Experience, pages 6
Issue 8-9, 2001. (http://www.globus.org/cog/java/)

[✓] I agree to these terms and conditions

You can either load a previous Java CoG Kit configuration, or start a new one.

If the second option is selected below, it means that a default previous configuration was not found.

The default configuration can usually be found in your .globus directory which is located in your home directory. The default name for the Java CoG Kit configuration file is "cog.properties".

(●) Use a previous configuration
/home/mike/.globus/cog.properties

Please enter the location of your Globus user certificate.

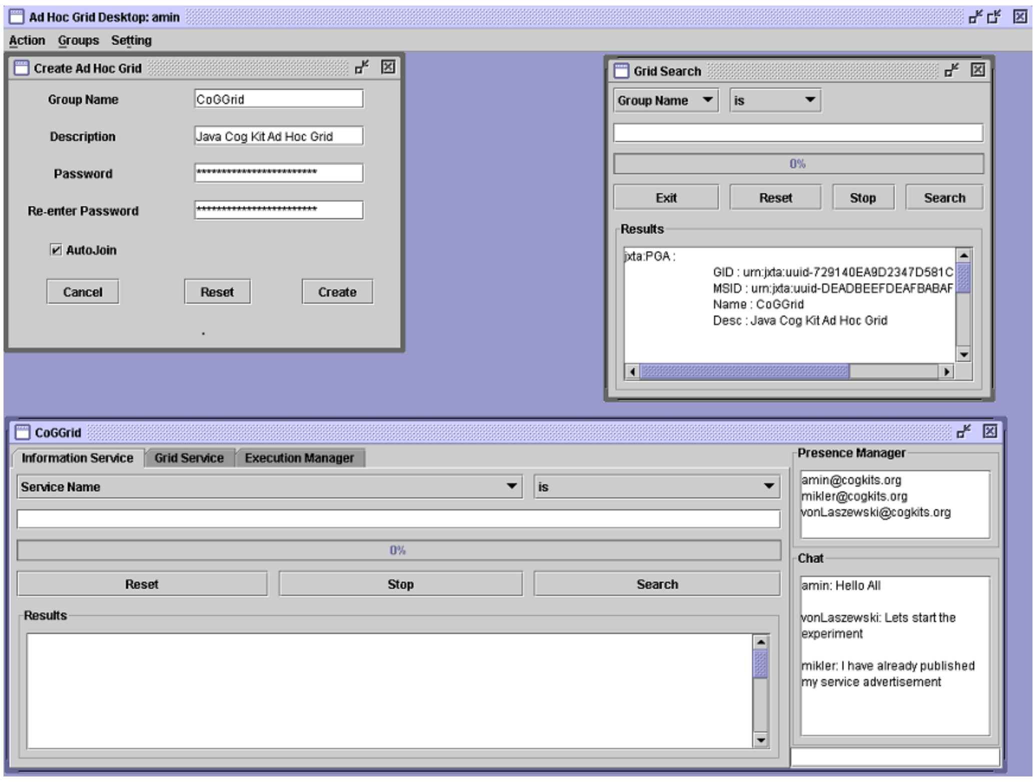
If you do not yet have a Globus certificate please consult the Globus manual on how to get one.

To browse for your Globus certificate please press the browse button ("...").

To view the selected file press "View".

/home/mike/.globus/usercert.pem

Fig. 5. The Java CoG Kit Grid environment setup component



Group Name: CoGGrid
Description: Java Cog Kit Ad Hoc Grid
Password: ************************
Re-enter Password: ************************
☑ AutoJoin

Group Name ▾ is ▾

0%

Results
```
jxta:PGA :
                GID : urn:jxta:uuid-729140EA9D2347D581C
                MSID : urn:jxta:uuid-DEADBEEFDEAFBABAF
                Name : CoGGrid
                Desc : Java Cog Kit Ad Hoc Grid
```

Service Name ▾ is ▾

0%

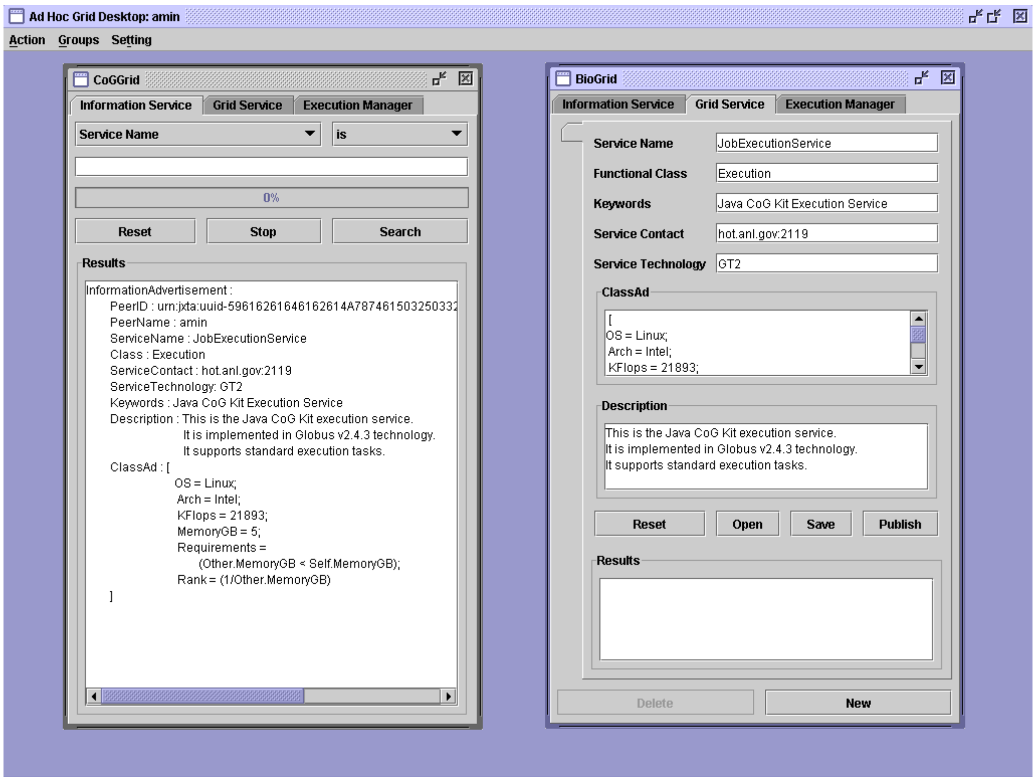
Presence Manager
amin@cogkits.org
mikler@cogkits.org
vonLaszewski@cogkits.org

Chat
amin: Hello All

vonLaszewski: Lets start the experiment

mikler: I have already published my service advertisement

Fig. 3. User interfaces for ad hoc Grid creation, discovery, presence management, and Grid-wide communication



Service Name ▾ is ▾

0%

Results
```
InformationAdvertisement :
     PeerID : urn:jxta:uuid-59616261646162614A787461503250332
     PeerName : amin
     ServiceName : JobExecutionService
     Class : Execution
     ServiceContact : hot.anl.gov:2119
     ServiceTechnology: GT2
     Keywords : Java CoG Kit Execution Service
     Description : This is the Java CoG Kit execution service.
                   It is implemented in Globus v2.4.3 technology.
                   It supports standard execution tasks.
     ClassAd : [
                  OS = Linux;
                  Arch = Intel;
                  KFlops = 21893;
                  MemoryGB = 5;
                  Requirements =
                       (Other.MemoryGB < Self.MemoryGB);
                  Rank = (1/Other.MemoryGB)
     ]
```

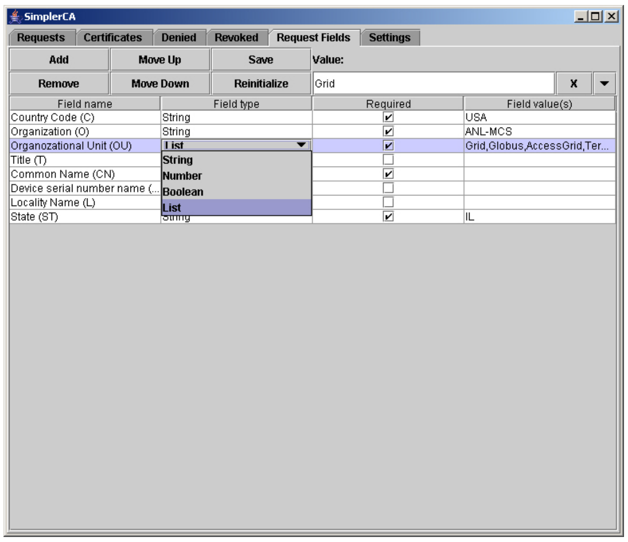
Service Name: JobExecutionService
Functional Class: Execution
Keywords: Java CoG Kit Execution Service
Service Contact: hot.anl.gov:2119
Service Technology: GT2

ClassAd
```
[
OS = Linux;
Arch = Intel;
KFlops = 21893;
```

Description
This is the Java CoG Kit execution service.
It is implemented in Globus v2.4.3 technology.
It supports standard execution tasks.

Fig. 4. User interfaces for publishing service advertisements and discovering published advertisements

Figure 5 shows the user interface for the Java CoG Kit setup component that allows ad hoc Grid peers to initialize their Grid environments, including the establishment of the trusted CA certificates. The authentication libraries utilized by AGSI are supplied by the *jglobus* module of the Java CoG Kit. The jglobus libraries are extensively used by the Grid community to facilitate its GSI requirements. The Java CoG Kit GSI module



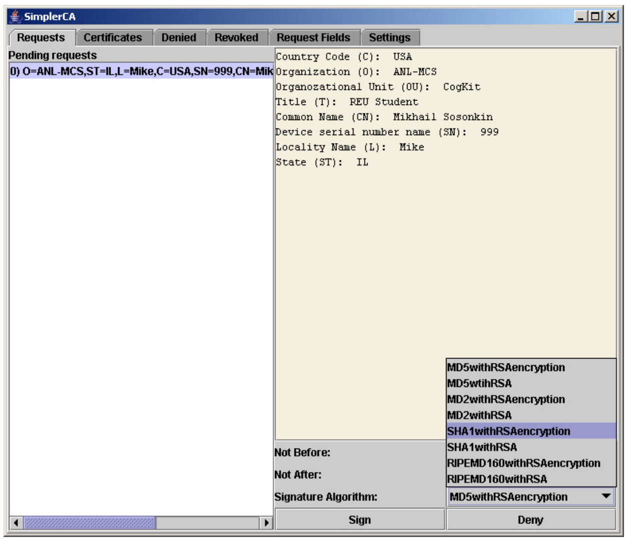
SimplerCA

Requests | Certificates | Denied | Revoked | Request Fields | Settings

Value: Grid

Field name	Field type	Required	Field value(s)
Country Code (C)	String	✓	USA
Organization (O)	String	✓	ANL-MCS
Organozational Unit (OU)	List	✓	Grid,Globus,AccessGrid,Ter...
Title (T)	String	☐	
Common Name (CN)	Number	✓	
Device serial number name (...	Boolean	☐	
Locality Name (L)	List	☐	
State (ST)	String	✓	IL

Fig. 6. SimplerCA interface for defining certificate attributes



SimplerCA

Requests | Certificates | Denied | Revoked | Request Fields | Settings

Pending requests

0) O=ANL-MCS,ST=IL,L=Mike,C=USA,SN=999,CN=Mik

```
Country Code (C):  USA
Organization (O):  ANL-MCS
Organozational Unit (OU):  CogKit
Title (T):  REU Student
Common Name (CN):  Mikhail Sosonkin
Device serial number name (SN):  999
Locality Name (L):  Mike
State (ST):  IL
```

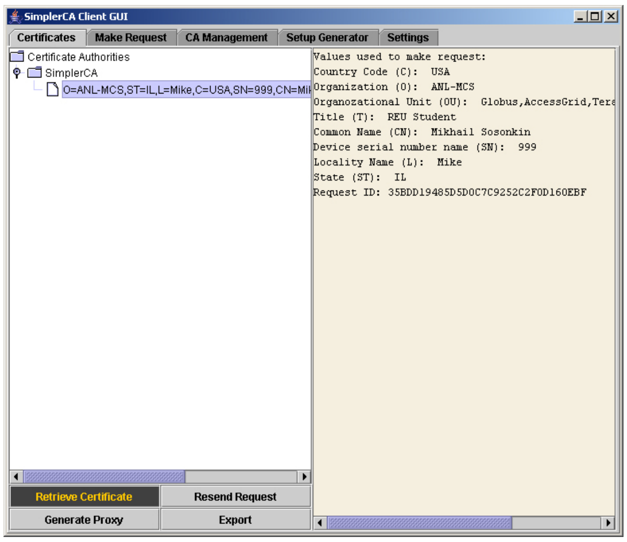
Not Before:

Not After:

Signature Algorithm: MD5withRSAencryption

MD5withRSAencryption
MD5withRSA
MD2withRSAencryption
MD2withRSA
SHA1withRSAencryption
SHA1withRSA
RIPEMD160withRSAencryption
RIPEMD160withRSA

Fig. 7. SimplerCA interface for approving certificate requests



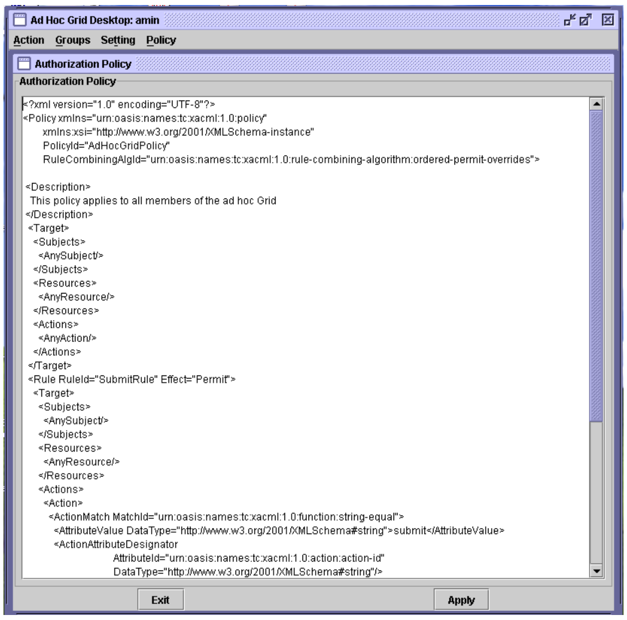
SimplerCA Client GUI

Certificates | Make Request | CA Management | Setup Generator | Settings

Certificate Authorities
SimplerCA
O=ANL-MCS,ST=IL,L=Mike,C=USA,SN=999,CN=Mi

```
Values used to make request:
Country Code (C):  USA
Organization (O):  ANL-MCS
Organozational Unit (OU):  Globus,AccessGrid,Tera
Title (T):  REU Student
Common Name (CN):  Mikhail Sosonkin
Device serial number name (SN):  999
Locality Name (L):  Mike
State (ST):  IL
Request ID: 35BDD19485D5D0C7C9252C2F0D160EBF
```

Fig. 8. SimplerCA client interface for requesting ad hoc Grid certificates



Ad Hoc Grid Desktop: amin

Action Groups Setting Policy

Authorization Policy

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    PolicyId="AdHocGridPolicy"
    RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:ordered-permit-overrides">

 <Description>
  This policy applies to all members of the ad hoc Grid
 </Description>
  <Target>
   <Subjects>
    <AnySubject/>
   </Subjects>
   <Resources>
    <AnyResource/>
   </Resources>
   <Actions>
    <AnyAction/>
   </Actions>
  </Target>
  <Rule RuleId="SubmitRule" Effect="Permit">
   <Target>
    <Subjects>
     <AnySubject/>
    </Subjects>
    <Resources>
     <AnyResource/>
    </Resources>
    <Actions>
     <Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">submit</AttributeValue>
       <ActionAttributeDesignator
                  AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                  DataType="http://www.w3.org/2001/XMLSchema#string"/>
```

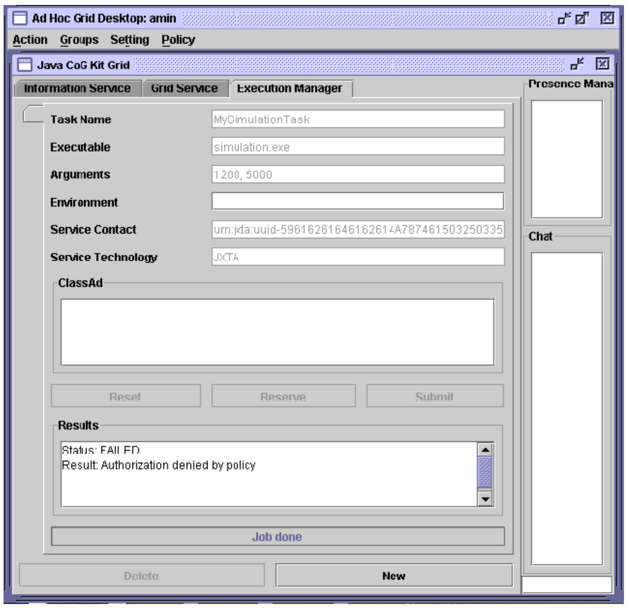
Fig. 9. User interface allowing the peer to express authorization policies

is also internally used and distributed with the Globus toolkit versions 3 and 4. The Java CoG Kit (cog-jglobus) provides a standards-based solution to the mutual authentication and single-sign-on problems. Further, to facilitate the creation and distribution of dynamic Grid identities, we have adopted the SimplerCA certificate management system from the Java CoG Kit [6]. Figure 6 shows the interface allowing CA peers to establish the necessary attributes required for identity generation. Figure 7 shows a snapshot of the outstanding certificate requests pending approval from the CA. Subject to its policy, the peer can approve or reject these requests, appropriately notifying the requester. Figure 8 shows the client-side management of CAs and their generated identities. It allows clients to add new CAs, send certificate requests, and generate proxy credentials for accepted certificates.

As discussed earlier, access to a contributed service is controlled by XACML authorization policies. Figure 9 shows the user interface allowing peers to express their authorization policies in XACML syntax. We acknowledge the fact that expressing XACML policies in textual format is cumbersome and not user-friendly. Ongoing activities are focusing on creating a flexible and user-friendly tool allowing peers to configure their policies with a graphical interface. The PEP and the PDP in our prototype is implemented using the open-source Java implementation of the XACML specification, called as sunxacml [20]. Figure 10 shows the user interface enabling peers to invoke access controlled community services. Access will be successful only if the requesting peer satisfies the authorization policy of the remote service.

V. Related Work

Extensive research has been conducted by the Grid community on community-based security infrastructures.



Task Name: MySimulationTask
Executable: simulation.exe
Arguments: 1200, 5000
Environment:
Service Contact: urn:jxta:uuid-59616261646162614A787461503250335
Service Technology: JXTA

Status: FAILED
Result: Authorization denied by policy

Fig. 10. User interface for submitting Grid tasks to access controlled community services

A majority of the proposed solutions focus on decoupling the Grid resource administration from community administration. Very few security frameworks discussed in literature focus on structure- and control-independent security solutions, the primary focus of AGSI. In this section we discuss some of the relevant security frameworks that seek to support community-controlled solutions.

The Community Authorization Service (CAS) [21] framework from the Globus Alliance [9] builds on the GSI-based authentication system to support fine-grained group authorization. It segregates the administration of resources from the administration of Grid communities. Every Grid community instantiates a CAS server representing that community and controlled by a community administrator. This administrator acquires coarse-grained authorization privileges from the resource administrator on behalf of the community. Within the scope of these privileges, the community administrator manages fine-grained authorization permissions among the community users based on the community-specific trust relationships. Using the CAS server, community members obtain their individual community privileges in the form of limited proxy credentials. With these restricted credentials, the community members can access a subset of resource functionality available to the community.

A community-based authorization framework can also be formed by using the Virtual Organization Membership Service (VOMS) [22]. Every virtual organization (VO) has an associated VOMS server and a VOMS administrator. Resource administrators grant bulk privileges to the VO at a coarse level. These privileges are distributed to the community members via the VOMS server using fine-grained trust relationships. The VOMS-based system differs from the CAS framework in its representation of the community privileges. While the CAS framework assigns community privileges as restricted X.509 proxy certificates, the VOMS-based system assigns them as privilege attribute certificates.

Although CAS and the VOMS-based system enable the Grid communities to manage their own fine-grained trust relationships, they differ from AGSI in their strong reliance on the CAS and VOMS servers, respectively. Further, their dependence on a pre-established community administrator prohibits them from supporting the structure- and control-independent requirements of ad hoc Grids. The requirements imposed by these systems in terms of its community-owned static-infrastructure components cannot be satisfied by ad hoc Grids.

The Akenti [23] system enforces access control on resources based on policies expressed by multiple authoritative entities (stakeholders). Multiple stakeholders for an Akenti-enforced resource can impose access control requirements independent of other stakeholders. Resource access is granted to users based on their identity credentials and the dynamically aggregated authorization policies from all the involved stakeholders.

The PRIMA [24] privilege management framework is conceptually similar to Akenti. It allows multiple entities that are authoritative for a resource to delegate access to resources for which they are authoritative. Users can possess and further delegate to other users fine-grained privileges to resources for which they are authoritative. Resource privileges are expressed and distributed as privilege attributes. Therefore, access to a resource enforced by PRIMA is based on the aggregate set of privilege attributes presented by the user.

The Akenti and PRIMA systems offer excellent decentralized solutions to distributed authorization schemes without requiring any community-owned static infrastructure. Nevertheless, they are fundamentally based on the assumptions of a hierarchical and multi-entity resource authoritative system, where a resource is controlled by multiple entities. While this assumption is valid for traditional Grid systems, it does not hold true for ad hoc Grids. In the ad hoc Grid model, every peer bears the exclusive responsibility for and control of the services contributed by it. In the absence of such multiple-authoritative requirements, the Akenti and PRIMA systems result in a heavy weight solution trying to solve problems that do not even occur in ad hoc Grids. The authorization subsystem offered in this paper is a light weight solution offering a subset of functionalities available in Akenti and PRIMA, specifically targeting the requirements and characteristics of ad hoc Grids.

VI. Summary

Security is one of the pillars of any Grid environment. This paper describes a security model capable of supporting the requirements imposed by ad hoc Grids: structure independence and control independence. We refer to the security framework described in this paper as the ad hoc Grid security infrastructure (AGSI). The authentication subsystem of AGSI is responsible for creating, managing, and verifying Grid identities in ad hoc Grids. It adopts the GSI model for mutual authentication and single sign-on. Unlike GSI, however, AGSI does not support credential delegation. AGSI also supports dynamic identity generation using the SimplerCA certificate management system. The authorization subsystem of AGSI is based on the enforcement of standards-based XACML access control policies. These policies allow every peer to autonomously configure and enforce access to services offered by it. To validate and verify our AGSI model, we integrated it with a prototype ad hoc Grid implementation. Our implementation allows ad hoc Grid peers to create, discover, and participate in spontaneous Grid collaborations. Peers can contribute autonomously protected services to the community. Grid identities are dynamically established by the peers allowing authenticated service interaction. Access to services are permitted only if the requester satisfies the authorization policies imposed by the service provider. Ongoing research activities within the Java CoG Kit project is focusing on integrating an incremental reputation system with AGSI.

ACKNOWLEDGMENTS

This work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-Eng-38. DARPA, DOE, and NSF support Globus Project research and development. The Java CoG Kit Project is supported by DOE MICS, and NSF Alliance.

REFERENCES

[1] G. von Laszewski, M.-H. Su, J. A. Insley, I. Foster, J. Bresnahan, C. Kesselman, M. Thiebaux, M. L. Rivers, S. Wang, B. Tieman, and I. McNulty, "Real-Time Analysis, Visualization, and Steering of Microtomography Experiments at Photon Sources," in *Ninth SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio, TX, 22-24 Mar. 1999. http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--siamCmt99.pdf

[2] G. von Laszewski, M. Westbrook, I. Foster, E. Westbrook, and C. Barnes, "Using Computational Grid Capabilities to Enhance the Ability of an X-Ray Source for Structural Biology," *Cluster Computing*, vol. 3, no. 3, pp. 187–199, 2000. http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--dtrek.pdf

[3] G. von Laszewski and P. Wagstrom, *Tools and Environments for Parallel and Distributed Computing*, ser. Parallel and Distributed Computing. Wiley, 2004, ch. Gestalt of the Grid, pp. 149–187. http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--gestalt.pdf

[4] G. von Laszewski, J. Gawor, C. J. Peña, and I. Foster, "InfoGram: A Peer-to-Peer Information and Job Submission Service," in *Proceedings of the 11th Symposium on High Performance Distributed Computing*, Edinbrough, U.K., 24-26 July 2002, pp. 333–342. http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--infogram.pdf

[5] G. von Laszewski, E. Blau, M. Bletzinger, J. Gawor, P. Lane, S. Martin, and M. Russell, "Software, Component, and Service Deployment in Computational Grids," in *IFIP/ACM Working Conference on Component Deployment*, ser. Lecture Notes in Computer Science, J. Bishop, Ed., vol. 2370. Berlin, Germany: Springer, 20-21 June 2002, pp. 244–256. http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--deploy-32.pdf

[6] G. von Laszewski and M. Sosonkin, "A Grid Certificate Authority for Community and Ad-hoc Grids," in *7th International Workshop on Java for Parallel and Distributed Computing, published in the Proceedings of the 19th International Parallel and Distributed Processing Symposium*. Denver, CO: IEEE, 4-8 Apr. 2005. http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-ca-workshop.pdf

[7] K. Amin, G. von Laszewski, and A. R. Mikler, "Grid Computing for the Masses: An Overview," in *Grid and Cooperative Computing (GCC2003)*, Shanghai, China, December 2003, pp. 464–473. http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--masses-gcc03.pdf

[8] G. von Laszewski, I. Foster, J. Gawor, and P. Lane, "A Java Commodity Grid Kit," *Concurrency and Computation: Practice and Experience*, vol. 13, no. 8-9, pp. 643–662, 2001. http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--cog-cpe-final.pdf

[9] "The Globus Alliance," Web Page. http://www.globus.org

[10] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," in *5th ACM Conference on Computer and Communications Security*. ACM Press, Nov. 2-5 1998, pp. 83–92. ftp://ftp.globus.org/pub/globus/papers/security.pdf

[11] R. Housley, W. Polk, W. Ford, and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 3280, April 2002.

[12] "SSL Certificates from VeriSign, Inc." http://www.verisign.com/products-services/security-services/ssl/index.html

[13] "DOEGrids Certificate Service," Web Page. http://www.doegrids.org/

[14] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, and F. Siebenlist, "X.509 Proxy Certificates for Dynamic Delegation," in *Third Annual PKI R & D Workshop*, 2004.

[15] "OASIS eXtensible Access Control Markup Language (XACML) TC." http://www.oasis-open.org/committees/xacml

[16] R. Yavatkar, D. Pendarakis, and R. Guerin, "A Framework for Policy-based Admission Control," RFC 2753, January 2000.

[17] S. Godik and T. Moses, "eXtensible Access Control Markup Language (XACML) Version 1.1," OASIS Standard, August 2003.

[18] "Project JXTA," Web Page. http://www.jxta.org/

[19] R. Raman, "Matchmaking Frameworks for Distributed Resource Management," Ph.D. dissertation, The University of Wisconsin-Madison, 2000.

[20] "Sun's XACML Implementation." http://sunxacml.sourceforge.net/

[21] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke, "A Community Authorization Service for Group Collaboration," in *IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, Monterey CA, USA, 5-7 June 2002.

[22] "VOMS Architecture v1.1," Web Page, May 2002. http://grid-auth.infn.it/docs/VOMS-v1_1.pdf

[23] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari, "Certificate-based Access Control for Widely Distributed Resources," 1999.

[24] M. Lorch and D. G. Kafura, "The PRIMA Grid Authorization System," *Journal of Grid Computing*, vol. 2, no. 3, pp. 279–298, 2004.